Università degli Studi di Milano - Bicocca

Dipartimento di Economia e Statistica

Corso di Laurea Triennale in Scienze Statistiche Economiche

Applicazione di Tecniche di Machine Learning per la Classificazione di Testi descrittivi delle attività Economiche

Relatore: Ing. Mirko Cesarini

Relazione della prova finale di:

Lombardi Mattia

Matricola 837193

Anno Accademico 2021-2022

Alla mia famiglia

Indice

Abstract

La Text Classification è diventata un importante ambito di ricerca ed applicazione da quando sono stati introdotti i documenti digitali.
Il vertiginoso aumento dei dati testuali nel web e nelle aziende ha reso difficoltoso il tradizionale metodo di classificazione manuale, che è diventato oneroso sia in termini di tempistiche che quantità.
Pertanto, l'automatizzazione della classificazione dei testi, mediante tecniche di Machine Learning, può fornire un efficace metodo di risoluzione.

In questo elaborato verranno descritte delle tecniche e degli algoritmi di classificazione per automatizzare il processo di catalogazione delle principali attività economiche dei clienti di un'organizzazione.

Capitolo 1

Introduzione

Il progetto di questo elaborato è stato ideato durante lo svolgimento del tirocinio curriculare presso **Hilti Italia**. Un'azienda che progetta e produce tecnologie, software e servizi a supporto dei professionisti che lavorano nel mondo dell'edilizia. E' possibile consultare il loro sito web al seguente indirizzo: `https://www.hilti.it/`

Uno degli obiettivi target dell'azienda è la fidelizzazione con i clienti, in modo tale da poter offrire strumenti e servizi di qualità nel modo più efficiente e tempestivo possibile.

Pertanto, una celere e corretta raccolta dei dati dei clienti è un fattore importante per il business, poiché in questo modo si può avere un'ampia visuale della propria area di mercato.

1.1 Background

Durante lo svolgimento dello stage sono stato inserito nel *team* "Data Quality", il cui compito principale era quello di gestire correttamente le anagrafiche dei clienti. Durante il tirocinio ho supportato la squadra nella raccolta e nel controllo dei dati, inoltre ho avuto modo di vivere un'esperienza in un contesto aziendale e di apprendere le sue relative dinamiche.

Per le attività lavorative svolte durante il mio tirocinio, e riguardanti il team "Data Quality", l'azienda si è servita principalmente di un database relazionale, sviluppato da SAP (*Systems, Applications and Products in data processing*). Inoltre, sono stati adoperati marginalmente i software Excel e MySQL, per il trattamento dei dati.

Uno dei compiti che era di mia competenza riguardava la raccolta dei dati di nuovi clienti, quest'ultimi venivano inviati da un venditore di Hilti oppure direttamente dal cliente tramite registrazione o altri canali. Prima di venir inseriti all'interno del database queste informazioni dovevano venir controllate e confrontate con una fonte ufficiale, per questo motivo Hilti si serve della piattaforma *Atoka* di Cerved. Quest'ultimo è un servizio che mette a disposizione i principali dati delle aziende, che vengono presi prevalentemente dalla visura camerale e, pertanto, sono di origine certificata. Infatti, la visura camerale è un documento che fornisce informazioni sull'impresa del cliente in questione e viene sancito al momento dell'iscrizione della società al registro delle imprese.
Alcuni dei dati che sono presenti nella visura camerale sono il codice ateco ed una descrizione delle attività principali che la società si propone di svolgere, altresì chiamato oggetto sociale.

Queste informazioni dovevano venir lette ed interpretate da una persona del *team*, in modo tale da attribuire la corretta codificazione interna delle

attività economiche dei clienti di Hilti. Infatti, l'azienda aveva un documento in cui venivano presentate le diverse tipologie di clienti suddivise per attività economica svolta.

Questa mansione risultava molto pesante e difficile da gestire, soprattutto quando vi erano una moltitudine di nuovi clienti. Pertanto, ho proposto di sviluppare e testare degli algoritmi di *Machine Learning* per far interpretare ai computer i testi delle descrizioni economiche e scegliere il codice più opportuno in maniera automatizzata.
In questo modo si potrebbero diminuire i costi relativi alla forza lavoro ed ottimizzare i tempi per l'acquisizione delle informazioni. Inoltre, gli algoritmi di ML se ben "allenati" e con dati di qualità commettono meno errori della controparte umana. Questo progetto di tesi è un punto di riferimento per implementare un sistema automatizzato per la raccolta dei dati riguardanti le attività economica svolte dalle imprese dei clienti.

Verranno presentante le tecniche e metodologie per tradurre il testo in variabili, in modo tale da poter successivamente essere inserite come *input* nei classificatori, i quali restituiranno come *output* il codice aziendale previsto, rappresentante l'attività economica svolta.

1.2 Strumenti

Per l'implementazione degli algoritmi e delle metodologie presentate in questo elaborato è stato principalmente adoperato il software **Rstudio®**, un ambiente di sviluppo integrato (IDE, *Integrated Development Enviromenti*), per programmare in linguaggio *R*.
Sono state adoperate diverse librerie e pacchetti a seconda del compito da svolgere:

- **tidyverse**, un pacchetto che serve per l'importazione, la manipolazione e la visualizzazione dei dati. Per quanto concerne la creazione di grafici personalizzabili si utilizza l'apposita libreria "ggplot2";
- **tm** e **quanteda**, sono i due pacchetti che sono stati adoperati per l'analisi ed il trattamento dei dati testuali;
- **lsa**, il cui acronimo sta ad indicare *Latent Semantic Analysis* ed è un pacchetto che è stato adoperato per calcolare la similarità tra documenti;
- **e1071** è un pacchetto che presenta numerose funzioni per implementare modelli ed algoritmi di apprendimento automatico, come i classificatori *Naive Bayes*, SVM e *Random Forest*.

Inoltre, è stato adoperato anche il software **MySQL**, un sistema di gestione open-source, che permette la progettazione di database relazionali e la loro interrogazione tramite *query* SQL. In particolar modo è servito per integrare due diverse fonti di dati per la creazione di un dataset finale da adoperare per l'apprendimento dei modelli.

Infine, per la scrittura di questo elaborato di tesi è stato utilizzato **LaTeX** [1] , un software per la preparazione e messa a punto di testi e documenti.

1.3 Struttura dell'Elaborato

Questo elaborato è composto da un corpo centrale, senza considerare l'introduzione e le conclusioni, di 5 capitoli:

- **Stato dell'arte**, in questo 2 capitolo verrà presentato il problema di classificazione nella sua forma più semplice, ovvero tramite una funzione. Inoltre, verranno presentate le principali tecniche utilizzate per compiti di classificazione del testo ed alcuni cenni di storia del *Machine Learning*;

- **Raccolta e Preparazione dei dati**, in questo 3 capitolo verranno raccolti e presentati i dati che serviranno per implementare i modelli di apprendimento. Verranno valutate le principali dimensioni riguardanti la qualità dei dati facendo particolare attenzione ai dati mancanti ed allo sbilanciamento delle classi;

- **Text Mining** è il 4 capitolo e qui verranno presentate le principali tecniche adottate di analisi e manipolazione del testo. L'obiettivo principale sarà quello di trasformare i documenti in un formato standardizzato e quindi interpretabile dai calcolatori;

- **Applicazione dei Modelli**, in questo 5 capitolo verranno presentati gli algoritmi di classificazione utilizzati e la loro messa a punto;

- **Risultati degli Esperimenti**, in questo 6 capitolo saranno ordinati i risultati ottenuti dai classificatori. Prima verranno presentante le $performance$ generali e poi si considereranno le singole classi.

Non verranno visualizzati dati di aziende terze per motivi di riservatezza, i dati riguardanti le attività economiche svolte dai clienti sono anonimi.

Capitolo 2

Stato dell'Arte

La classificazione testuale si riferisce al processo di suddivisione di testi, che presentano caratteristiche simili, all'interno della medesima categoria.
Il fine è quello di mappare, proprio come farebbe una funzione matematica, gli oggetti in una determinata classe. Si considerino i dati che devono essere classificati come l'insieme dei documenti $\mathbf{D} = \{d_1, \ldots, d_n\}$, dove n rappresenta il numero totale di documenti. Le classi, invece, sono rappresentate dall'insieme $\mathbf{C} = \{c_1, \ldots, c_m\}$, dove m sono il numero di categorie predefinite.

$$f : D \rightarrow C$$

In questa applicazione l'insieme D è rappresentato dall'insieme di documenti testuali contenenti la descrizione ateco e l'oggetto sociale, mentre le classi C consistono nelle prime 5 categorie di clienti dell'azienda in cui ho svolto il tirocinio (presentante nel Paragrafo 3.5).
La funzione f invece si riferisce al compito di classificazione, quest'ultimo può essere svolto manualmente oppure mediante tecniche di *Text Classification* [2].

Esistono 3 principali approcci:

1. **Metodi basati su regole** consistono nel definire un insieme di regole (*if-then*) per determinare quale categoria dovrebbe essere assegnata ad un determinato documento testuale. Ad esempio, se una mail contiene certi slogan o proviene da un determinato interlocutore può venir automaticamente catalogata come *spam*. Questi metodi possono risultare efficienti per semplici compiti di classificazione o per completare altri modelli di ML, ma risultano difficili da manutenere ed aggiornare all'aumentare delle categorie e della complessità dei testi;

2. **Modelli di Machine Learning** suddivisi principalmente in:

   - *Supervised Learning*, abbiamo a disposizione dei dati la cui variabile predittiva risulta già classificata, pertanto i modelli potranno apprendere dai dati;
   - *Unsupervised Learning*, in tal caso non abbiamo a disposizione la variabile predittiva già etichettata. Risultano meno efficaci dei metodi supervisionati, ma possono servire per trovare *pattern* e raggruppare i dati. Inoltre, hanno il vantaggio che possono essere impiegati anche se i dati non sono ancora stati etichettati.

3. **Metodi Ibridi** combinano diversi approcci sia basati su regole che su modelli di ML. Possono essere adoperati per migliorare le $performance$ sfruttando al meglio i punti di forza dei diversi classificatori.

Le prime idee e proposte riguardanti l'applicazione di tecniche automatiche per la *Text Classification* (TC) sono state sviluppate a partire dagli anni 60'. In particolare, alla Cornell University da Salton è stato sviluppato lo *SMART Information Retrieval System* (SIRS) [3]. Tale sistema è stato migliorato negli anni a seguire ed ha contribuito alla

scoperta di diversi metodi ed algoritmi innovativi che vengono tutt'ora adoperati nell'ambito dell'*Information Retrieval* (IR), tra cui anche i modelli *Vector-Space* (VSM) [4] e metodi di pianificazione del testo come il *tf/idf* [5] che verranno successivamente discussi ed implementati nel corso di questo elaborato.

La *Text Classification* è una tecnica che fa parte del *Natural Language Processing* (NLP), una branca della scienza che ha come obiettivo quello di permettere ai computer di comprendere ed interpretare il linguaggio naturale. Il *Natural Language Processing* è stato adoperato anche per compiti di: traduzione e correzione di testi, rilevamento di spam nelle mail, sintesi, estrazione delle informazioni dai testi e tanti altri ancora.

2.1 Cenni di storia del Machine Learning

Il *Machine Learning* è una disciplina che si è sviluppata a partire dagli anni 1940-1950, ma che ha continuato ad evolversi (e si sta ancora evolvendo) fino ai giorni nostri. Pertanto, è il risultato collettivo di molti anni, menti e dottrine. La definizione formale di ML invece è stata data per la prima volta da Arthur Samuel nel 1959 [6], che cita:

> "Field of study that gives computers the ability to learn without being explicitly programmed"

Tra i precursori del ML abbiamo Walter Pitts (matematico) e Warren McCulloch (neuroscienziato) che nel 1943 hanno ideato il primo modello matematico di una rete neurale [7]. Il concetto teorico di *Artificial Intelligence* invece viene generalmente attribuito al noto matematico Alan Turing con la sua "Macchina di Turing" (1936) [8], una macchina ideale che fosse in grado di eseguire ogni tipo di calcolo su numeri e simboli.

Tuttavia, anche se le radici del ML sono state piantate negli anni 50' bisognerà aspettare l'inizio degli anni Novanta per vedere i primi frutti delle applicazioni. Infatti, inizialmente non si avevano a disposizione abbastanza dati e potenza computazionale per addestrare a dovere i modelli. La prima "vittoria" dell'AI è stata ottenuta nel 1997 quando *Deep Blue* [9], il supercomputer dell'IBM, ha battuto il campione in carica di scacchi Garry Kasparov.

Ad inizio anni 2000 l'interesse è stato rivolto verso il *Deep Learning* (DL), che si sviluppa nel 2006 grazie alla pubblicazione "A Fast Learning Algorithm for Deep Belief Nets" di Geoffrey Hinton [10]. Il *Deep Learning* è un campo di applicazione del ML che adopera algoritmi (come ad esempio CNNs [11] e RNNs [12]) che sono ispirati alla struttura e funzioni delle reti neurali che compongono il cervello. Negli anni successivi tali modelli hanno raggiunto lo stato dell'arte in diversi campi ed applicazioni come: "DeepFace" (2014) [13],"AlphaGo" (2016) [14].

Per quanto riguarda l'ambito della NLP c'è stato un enorme passo avanti di recente con l'annuncio di ChatGPT-3, prodotta e presentata a fine 2022 dall'azienda di ricerca OpenAI. Quest'ultima applicazione ha raggiunto lo stato dell'arte per quanto riguarda la quasi totalità di compiti riguardanti il linguaggio naturale. È stata costruita tramite reti neurali e dei metodi di apprendimento come il *reinforcement learning* ed il *transfer learning*, una tecnica di apprendimento automatico che consiste nell'utilizzare un modello pre-addestrato su una grande quantità di dati come base di *training* per gestire nuove situazioni in maniera accurata, anche con pochi dati a disposizione.
In questo elaborato non si farà riferimento al *Deep Learning* (DL), ma ai metodi classici del ML come il classificatore *Naive* di Bayes [15], SVM [16] e *Random Forest* [17] ed alle relative tecniche usate per l'interpretazione del testo in linguaggio informatico ed alla preparazione e standar-

dizzazione delle variabili.

Inoltre, è bene considerare che anche se i modelli basati sulle reti neurali hanno acquisito lo stato dell'arte nel NLP necessitano di molti più dati e tempo di allenamento. Pertanto, prima di implementare modelli di questo genere è una buona pratica creare dei modelli utilizzando classificatori con tempi di addestramento più ridotti.

Capitolo 3

Raccolta e Preparazione Dati

Le aziende negli ultimi anni hanno generato una quantità sempre maggiore di dati. Molte di queste informazioni però rimangono inutilizzate e, di conseguenza, non vengono sfruttate opportunità di analisi.
Per questo motivo il corretto raccoglimento dei dati di un'organizzazione e la loro successiva integrazione e lavorazione rappresentano degli step fondamentali per fornire una visione completa e cogliere opportunità di business. Inoltre in questo modo è possibile cercare pattern significativi nei dati, procedimento che prende anche il nome di *Knowledge Discovery in Databases* (KDD) [18].

3.1 Integrazione Dati

Il dataset per l'apprendimento dei modelli è stato ricavato mediante l'integrazione di due fonti di dati:

- il *Data Lake* di Hilti Italia, principale ambiente di archiviazione dei dati aziendali dell'organizzazione in cui ho svolto l'attività di stage. In questo archivio sono depositate le informazioni principali dei clienti dell'azienda (codice identificativo, ragione sociale, forma giuridica, etc.) ;
- un *Silos* di dati proveniente dalla piattaforma Cerved Atoka, la piattaforma di cui si serve l'azienda per visualizzare i dati ufficiali

delle imprese dei clienti. Per questo progetto i dati di interesse saranno, per l'appunto, la descrizione economica dell'attività ed il codice ateco. La piattaforma distribuisce all'azienda dati qualificati dei clienti ogni 3 mesi.

I seguenti dataset sono stati integrati mediante una *query* in linguaggio **SQL**, dove la chiave principale (in inglese *primary key*) per unire i dati delle diverse tabelle coinvolte è stata la P.IVA, che per sua natura è un codice univoco, dei clienti in oggetto.

```
SELECT [Customer Number 1]
      , [Name 1]
      , [Trade Code Level 4]
      , CODICE_ATECO_PRIMARIO
      , DESCRIZIONE_CODICE_ATECO
      , OGGETTO_SOCIALE
FROM TD_BASE_DATI_ATOKA_MASSIVA AS ATOKA
INNER JOIN
core.Customers_P72 AS C
ON RIFERIMENTO=[Customer Number 1]
```

Figura 3.1: *Codice della query che unisce i due dataset*

3.2 Struttura del Dataset

Dalla *query* rappresentata in Fig. 3.1 è stato ottenuto il dataset "Clienti Hilti ATOKA.xlsx". Quest'ultimo comprende 209.831 osservazioni delle seguenti variabili:

- **codice TRADE**, codificazione interna aziendale per identificare la tipologia di attività economica svolta dai clienti (factor);
- **codice cliente**, numero identificativo del cliente (numeric);
- **ragione sociale**, nome scelto dalla società (stringr);
- **codice ateco**, rappresenta la classificazione delle attività economiche adoperata da ISTAT [19] (factor);
- **descrizione del codice ateco**, una breve frase che descrive la principale attività economica della società (stringr);
- **oggetto sociale**, documento descrittivo rappresentante le attività che un'azienda si propone di svolgere al momento dell'atto costitutivo societario (stringr).

Il codice TRADE è stata considerata come variabile target, ovvero è la variabile che i classificatori cercano di predire a partire dai valori delle altre variabili. Questa cifratura differisce dalla codificazione ateco, quest'ultima rappresenta una tipologia di classificazione delle attività economiche che può avere fino a 6 cifre (quando presenta tutte le cifre viene detto completo). Le prime due cifre indicano il macro-settore dell'attività economica svolta, mentre le cifre successive rappresentano, con diversi gradi di dettaglio, le articolazioni e le disgregazioni dei diversi settori [19]. Pertanto i codici ateco sono molti più diversificati e numerosi dei codici Trade che sono stati scelti dall'azienda. Inoltre ogni organizzazione può decidere come classificare i clienti in base alla propria specifica segmentazione di mercato.

3.3 Data Cleansing

Una fase fondamentale che precede l'analisi dei dati e la loro successiva modellazione è la preparazione dei dati, che in inglese prende anche il nome di *Data Cleansing* (o *Cleaning*). Durante questa fase bisogna assicurarsi che i dati che andiamo ad utilizzare siano coerenti con le 6 seguenti dimensioni (presentate nel [20], capitolo 13):
completezza, unicità, tempestività, validità, accuratezza e consistenza.

Secondo le ricerche svolte da Gartner [21] la qualità dei dati di un'organizzazione è direttamente responsabile del valore del business.

D'altronde come afferma la nota espressione *"Garbage In, Garbage Out"* (GIGO), ossia se i dati in entrata sono scadenti si otterrano risultati scadenti, in quanto si prenderanno decisioni fallimentari.

3.3.1 Missing Values

Una delle problematiche più comuni a cui possiamo andare incontro durante la preparazione del nostro dataset è la presenza di dati mancanti (*missing values*), anche chiamati NA, ovvero *"Not Available"*.
Inoltre la probabilità che i dati siano mancanti aumenta con la dimensioni del dataset. I dati non disponibili possono essere:

- **MCAR** (*Missing Completely At Random*), valori mancanti completamente casuali. In tal caso, i valori mancanti non dipendono né dalla variabile stessa né da nessun'altra variabile presente nella tabella dati. Questa situazione è molto raro che si verifichi in dataset reali, ma è l'unica che può essere testata;

- **MAR** (*Missing At Random*), valori mancanti casuali. I valori mancanti sono indipendenti dal valore che viene a mancare, ma possono dipendere da altre variabili. Per esempio i dati mancanti sono MAR se in un questionario per diagnosticare la depressione i partecipanti maschi sono meno disposti a completare il test [22];

- **NMAR** (*Not Missing At Random*) valori mancanti non ignorabili, che dipendono dal valore del dato stesso. Per estendere esempio precedente i dati sono NMAR se, dato un questionario per la depressione, i soggetti che sono gravemente depressi si rifiutano di fare il test [22].

Nel primo e secondo caso i dati mancanti possono essere ignorati (*case deletion*), anche se non è la migliore strategia, in alcuni casi non ci sono alternative. Una soluzione può essere l'integrazione con un' altro eventuale dataset che presenta dati più aggiornati e completi.

In questo caso sono stati analizzati i dati mancanti tramite la **library (visdat)** di **Rstudio®**. Particolarmente utile per la visualizzazione dei dati mancanti.

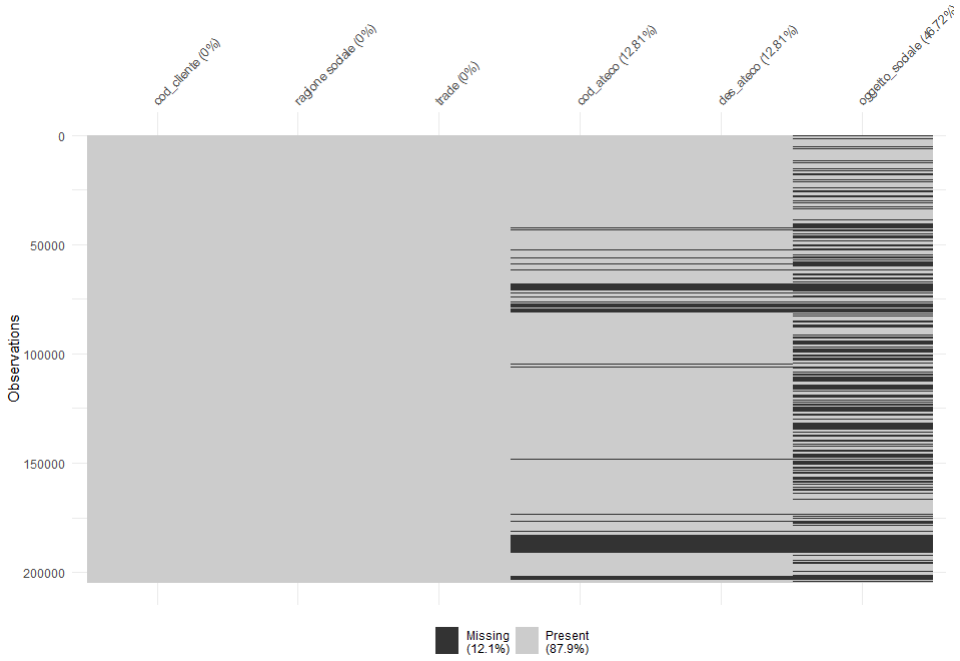

Figura 3.2: *Grafico rappresentante i dati mancanti nel dataset*

Come possiamo vedere in Fig. 3.2 i dati mancanti provengono principalmente dalle variabili "codice ateco" ed "oggetto sociale". Nel grafico i dati mancanti sono stati ordinati per codice Trade e possiamo notare

visivamente almeno un *pattern*. Pertanto probabilmente i dati mancanti non sono MCAR. Per poterlo valutare con certezza si può adottare il test di Little's (1988), presente nella **library (naniar)**. L'ipotesi nulla del test è che i valori mancanti siano MCAR ed adopera una statistica test distribuita come un Chi-quadro. Dato che il test può essere utilizzato unicamente con valori numerici (o factor) è stata considerata la relazione tra codici ateco e Trade: si ottiene così un *pvalue* prossimo allo 0 e due *pattern*, pertanto si può rifiutare l'ipotesi nulla che i dati mancanti siano MCAR. Possiamo vedere il risultato del test in Fig. 3.3 :

```
> library(naniar)
> mcar_test(atoka_test2)
# A tibble: 1 x 4
  statistic    df p.value missing.patterns
      <dbl> <dbl>   <dbl>            <int>
1      583.     1       0                2
```

Figura 3.3: *Codice rappresentante il risultato del test Little's*

Poiché abbiamo ottenuto due pattern tra i codici ateco e Trade si ipotizza che i dati mancanti siano MAR, ossia alcuni codici ateco sono più inclini a presentare dati mancanti. Inoltre quest'ultimi possono essere dovuti a variabili latenti, come il fatturato delle aziende o il numero di dipendenti. Dato che questo progetto riguarda la *Text Classification*, per i dataset di *training* e *testing* si è deciso di omettere i valori mancanti, poiché le descrizioni testuali risultano fondamentali per l'allenamento dei classificatori.

Bisogna però considerare che sarebbe opportuno richiedere alla piattaforma ATOKA (o ricercare tramite un'API personalizzata) altri dati testuali riguardanti l'attività economica delle aziende, come ad esempio la descrizione proveniente dal website. In questo modo potremmo integrare i nuovi dati nel dataset, limitare i dati mancanti ed avere più informazioni dei clienti a disposizione.

3.3.2 Consistenza

Un'altra importante dimensione a cui fare riferimento è la consistenza. Quest'ultima è definita in letteratura come la "violazione di una o più regole definite su un insieme di dati" [23].

Per esempio Hilti Italia concorda ogni anno quali siano i codici "trade" da mantenere attivi e se aggiungerne degli altri. Queste informazioni sono presenti nel file "Normativa TRADE 2022.xls". Durante il tirocinio avevo la responsabilità di consultare questo documento per poter attribuire il codice Trade, che ritenevo più adeguato, ai nuovi clienti.
Questo file verrà usato anche successivamente per calcolare la similarità tra testi nel Paragrafo 5.1.1.

Attraverso l'uso della **library (tidyverse)** i nostri dati sono stati manipolati in modo tale da omettere i codici "trade" che non risultavano più in vigore quest'anno. Da questa operazione sono stati trovati 4889 clienti che ancora riportavano un valore errato, secondo le normative attuali.

3.4 Sbilanciamento Classi

Un'altra problematica che in questo caso è attinente ai modelli di classificazione *supervised* è la *class imbalance*. Quest'ultima situazione si verifica quando la variabile predittiva Y presenta uno sbilanciamento più o meno marcato nella sua distribuzione categorica. Questo problema si presenta in maniera molto marcata in alcuni ambiti, come ad esempio nei rilevamenti delle frodi o di alcuni tumori maligni. In quest'ultimo caso si tratta di eventi sfavorevoli rari di cui si hanno a disposizione poche osservazioni.

Alcune tra le principali strategie per risolvere il problema:

- **Metodi di Livellamento**, i quali includono il *Random Under-Sampling* (RUS) ed il *Random Over-Sampling* (ROS). Come possiamo intuire dal nome, sono metodi che considerano casualmente meno (o più) osservazioni della classe maggioritaria (o minoritaria);

- ***Data Augmentation***, la principale tecnica è stata introdotta da Chawla con la *Syntethic Minority Over-Sampling Technique* (SMOTE) [24]. Tale metodo cerca di bilanciare le classi aumentando la classe minoritaria aggiungendo dei dati replicati casuali.;

- ***Cost-Sensitive Training***, si tratta di algoritmi che penalizzano il costo di classificazione per la classe maggioritaria;

- **Combinare classi minoritarie**, accorpare le classi minoritarie ad altre classi con caratteristiche simili;

- **Altre metriche di performance**, come ad esempio l'*Area Under ROC Curve* (AUROC), si tratta di una metrica che è meno suscettibile alla class-imbalance, in quanto considera il trade-off tra *True Positive Rate* (TPR) e *False Positive Rate* (FPR).

Queste sono alcune tra le principali tecniche per superare il problema della *class imbalance*, in quanto ne esistono molte altre e sono metodologie in continua evoluzione.

Per valutare la situazione di sbilanciamento relativa al nostro dataset possiamo servirci di un grafico. A tal proposito è stato realizzato un *barplot* orizzontale rappresentante la distribuzione di frequenze relative dei codici *trade*.

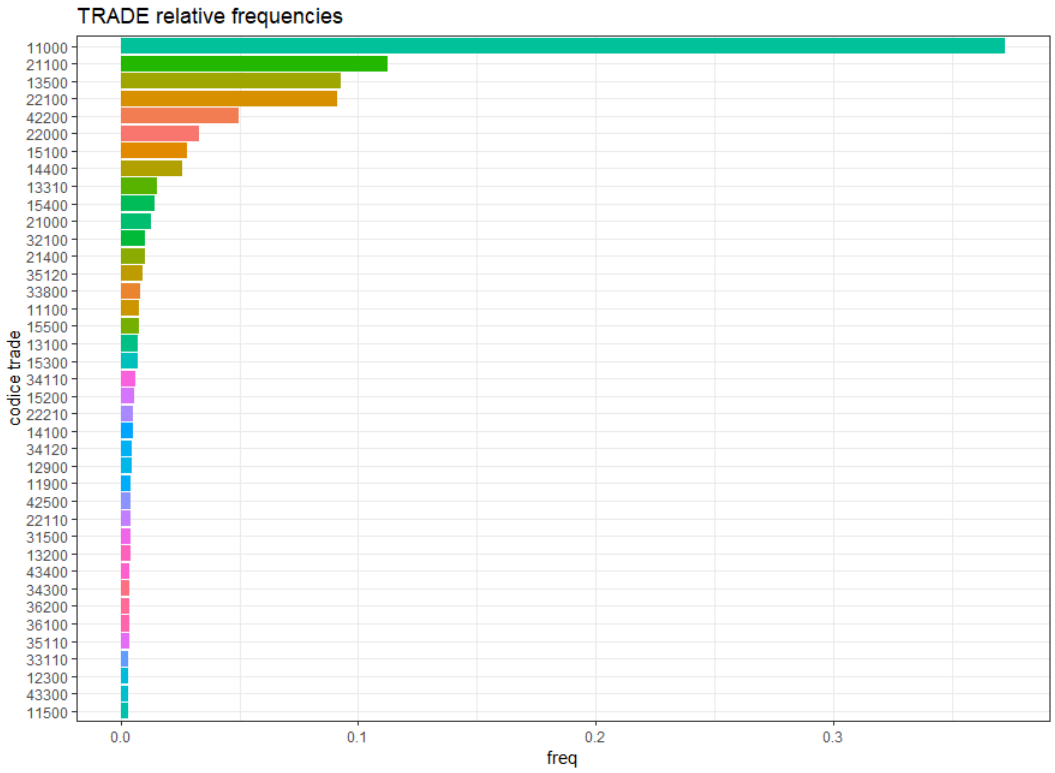

Figura 3.4: *Barplot che rappresenta distr. freq. rel. dei codici trade*

Come possiamo notare in Fig.3.4 i clienti presentano una classe maggioritaria, ossia il codice *trade* "11000", quest'ultima rappresenta il 37.25% del totale dei clienti di Hilti Italia. Il problema verrà affrontato tramite un *Random Under-Sampling* (RUS), presentato precedentemente, in modo tale da equiparare numericamente la classe maggioritaria alle altre. Quest'ultima tecnica potrebbe destare problemi quando non si hanno molti

dati a disposizione, poiché andiamo a ridurre la dimensione del dataset. Nel nostro caso, invece, non ha dato problemi in quanto avevamo molte osservazioni. Un'altra problematica che possiamo notare è la presenza di molte classi minoritarie, ossia tutte le categorie che presentano meno di 1000 osservazioni. Tali categorie non potranno beneficiare dell'etichettatura tramite *Machine Learning* e, pertanto, dovranno continuare ad essere gestite manualmente. Il processo di automatizzazione può essere ampliato a tutte le classi che presentano sufficienti osservazioni. Pertanto potrebbe risultare utile combinare alcune delle classi minoritarie tra loro.

```
73 library(ggplot2)
74 library(ggthemes)
75 library(RColorBrewer)
76
77 #we need to create a discrete color palette of the same lenght of the classes
78 colourCount=length(cont_trade$trade)
79 getPalette = colorRampPalette(brewer.pal(9, "Paired")) #reiterate colors
80
81 #graphic of the barplot
82 p<-ggplot(data=cont_trade, aes(x=reorder(trade,cont), y=cont/sum(cont),fill
   =getPalette(colourCount) )) +
83   geom_bar(stat="identity")+
84   labs(title ="TRADE relative frequencies",x="codice trade",y="freq")+
85   theme_bw()+
86   theme(legend.position = "none")
87
88 # Horizontal
89 p + coord_flip()
```

Figura 3.5: *Codice per creare il barplot soprastante*

3.5 Scelta delle Variabili

Quest'applicazione si concentrerà nella classificazione delle prime 5 attività economiche svolte dai clienti, le quali ricoprono il 72.85% dei clienti totali dell'azienda. I codici *trade* che verranno considerati come variabile predittiva Y sono:

- **Imprese Edili** (11000), edilizia per la costruzione di edifici;
- **Idraulica e Riscaldamento** (21100), imprese specializzate nell'installazione di impianti idraulici;
- **Elettricisti** (22100), imprese specializzate nell'installazione di impianti elettrici nel settore residenziali, commerciale ed industriale;
- **Fabbri e carpenterie leggere** (13500), imprese artigianali specializzate nella produzione ed installazione di strutture metalliche;
- **Altri Servizi commerciali** (42200), imprese che effettuano la vendita al dettaglio ed all'ingrosso (come ad esempio: supermarket, bar, centri sportivi, cinema etc...).

Si tratta pertanto di un'applicazione di *Multi Class Text Classification*, ossia un metodo di classificazione che prevede più di due classi, ma ogni *input* potrà avere un solo ed unico *output* (corrispondenza biunivoca). Le variabili esplicative saranno il codice ateco e le relative descrizioni, rispettivamente fornite da fonte ISTAT e dal cliente stesso nell'oggetto sociale. La ragione sociale non verrà considerata, poiché molte imprese come nome dell'azienda adoperano il proprio nominativo o un nome di fantasia, questo potrebbe generare troppo rumore nei dati. Vale a dire che sarebbero presenti molte più variabili che non sono rilevanti.

Al termine del *Random Under-Sampling* sono rimaste 38.559 osservazioni distribuite nelle 5 classi presentate come segue: "11000" - 0.04%, "13500" - 0.043%, "21100" - 0.04%, "22100" -0.035, "42200" - 0.03% .

Possiamo visualizzare la distribuzione in frequenza assoluta delle classi nel grafico di Fig. 3.6 :

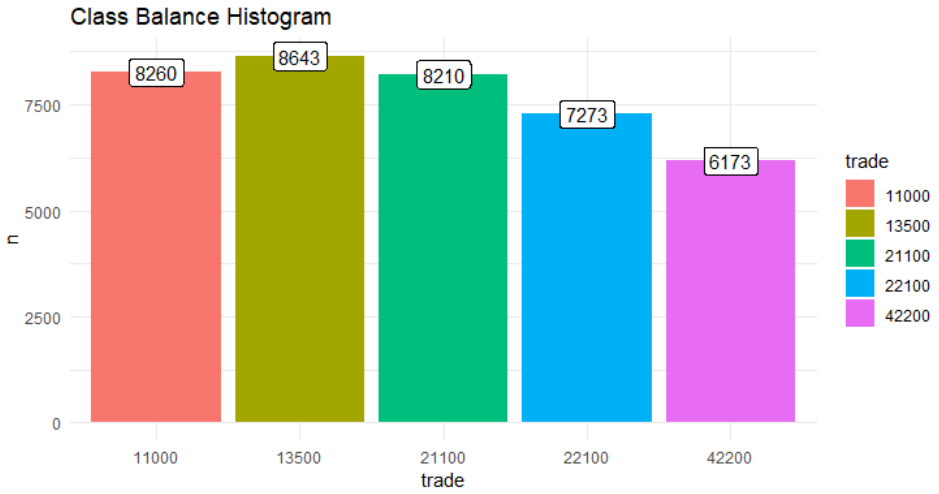

Figura 3.6: *Istogramma classi bilanciate*

Capitolo 4

Text Mining

I dati testuali rappresentano i principali veicoli di informazioni, secondo le stime presentate da Merill Lynch [25] più dell'80% di tutti i dati potenzialmente utili delle organizzazioni sono composte da dati non strutturati. Pertanto, avere degli strumenti e metodologie in grado di catturare queste informazioni è diventato un obiettivo per diverse tipologie di business.

Il text mining comprende un insieme di procedure in grado di trasformare il testo, non strutturato o semi strutturato, in un formato standardizzato e predisposto all'analisi o l'applicazione di algoritmi di machine learning.

Il linguaggio umano è molto variegato e presenta alcune ambiguità al suo interno. Ad esempio i sinonimi, termini diversi per indicare lo stesso concetto, o la polisemia, ossia parole che hanno significati diversi in base al contesto in cui sono inserite. Per questo motivo non è semplice far comprendere il significato dei testi alle macchine e bisogna filtrare ed organizzare i dati in modo tale che possano venir correttamente interpretati dai classificatori.

4.1 Standardizzazione del Testo

Uno step fondamentale è la preparazione dei dati testuali, infatti prima di venir processato il testo deve venir "pulito", procedimento che prende anche il nome di *text cleaning*. In questa fase il testo viene standardizzato mediante l'applicazione di diverse tecniche, di cui verrà fatta menzione nei seguenti sottocapitoli. Verranno presentati i principali metodi che sono stati utilizzati, ne esistono molti altri.
Sono state scelte queste tecniche in quanto sono state create ed adoperate con successo nel corso degli anni nell'ambito dell'IR (*Information Retrieval*. Pertanto, tali metodologie ben si prestano in questo ambito di applicazione. Infatti, è bene considerare quali tecniche possano rilevarsi più o meno adeguate per il target di interesse. Ad esempio, se l'obiettivo è quello di analizzare dati provenienti dai social media, potrebbe risultare utile trasformare le *emoticons* (simboli che rappresentano immagini nei messaggi) in testo, in modo tale da poterle processare.

```
82
83  library(tm)
84  library(SnowballC)
85  #pipeline for text cleaning
86  stemm = function(info){
87    info <- gsub("[^[:alnum:]]+", " ", info)
88    info <- tolower(info)
89    info <- removeNumbers(info)
90    info <- removeWords(info,stopwords('it'))
91    info <- stemDocument(info,language = "it")
92    return(info)
93  }
```

Figura 4.1: *Pipeline per la standardizzazione del testo*

Le seguenti operazioni di pulizia del testo sono state applicate nella *Pipeline* presente in Fig. 4.1. Con *Pipeline* nel ML si intende letteralmente una "conduttura per dati", ossia una sequenza di operazioni per il processing del dati, in cui l'output dell'operazione precedente diventa l'input dell'operazione successiva. La *Pipeline* viene principalmente

adottata per automatizzare alcuni step per la preparazione dei dati o la creazione di modelli.
Verranno ora presentate alcune componenti della pipeline, presente in Fig. 4.1.

4.1.1 Tokenization

La *Tokenization* è una tecnica che si occupa di suddividere il testo in *token* (i.e., blocco di testo atomico, tipicamente formato da caratteri indivisibili). Dal punto di vista applicativo si può definire il *token* come una qualsiasi sequenza di caratteri delimitata.

In base alle esigenze dell'analisi si può scegliere se ricercare parole, frasi o anche sequenze specifiche di lettere. In questo lavoro si è scelto di considerare come *token* le singole parole (*unigrammi*) ed una sequenza composta da due termini adiacenti (*bigrammi*). Si potrebbero considerare anche "n" parole ed in questo caso si parlerebbe di n-grammi.
Quest'ultima tecnica permette di includere più parole e, pertanto, comprendere meglio il senso del testo. Infatti, la considerazione di un solo termine potrebbe risultare fallace nella classificazione. Ad esempio, in questo campo di applicazione si potrebbero avere delle descrizioni con le due espressioni seguenti: sia "impianti elettrici" che "impianti idrici", in tal caso è bene che la parola "impianti" venga affiancata al termine successivo per comprendere di cosa si tratta nello specifico.
La scelta di considerare i termini fino ai bigrammi è stata fatta per due principali motivi: innanzitutto considerare più parole consecutive significherebbe avere più variabili nel modello e, quindi, una maggiore complessità computazionale. In secondo luogo, gli oggetti sociali e le descrizioni dei codici ateco sono pressoché sintetiche ed i bigrammi risultano sufficienti ad estrapolarne i concetti principali.

Inizialmente, si potrebbe pensare che gli spazi siano sufficienti a suddividere il testo, ma esistono numerose eccezioni. Ad esempio, l'apostrofo

di norma è situato in mezzo a due termini diversi che, se non altrimenti specificato, verrebbero erroneamente rappresentati come un unico *token*. Altri caratteri che possono essere considerati come delimitatori sono la punteggiatura o caratteri speciali. Infatti, questi sono stati considerati come divisori e sono stati sostituiti da uno spazio vuoto. In questo modo, quando il testo è stato suddiviso in termini non sono state considerati.
Inoltre, tutte le parole verranno considerate con lettere minuscole, ciò viene fatto per standardizzare i termini ed evitare che la stessa parola, quando rappresentata con iniziale maiuscola venga considerata come un altro termine.

4.1.2 Stopwords

Il concetto di *stopwords* è stato introdotto nel 1960 dal ricercatore e pioniere dell'*Information Retrieval* (IR) Hans Peter Luhn [26].
Le *stopwords* sono parole molto frequenti (come ad esempio articoli e preposizioni) che servono per connettere i termini di una frase, ma non aggiungono significato. Nell'ambito dell'IR queste parole vengono generalmente omesse e sono presenti numerose documentazioni a favore di questo metodo [27][28]. Inoltre, tale tecnica viene adoperata anche dai più comuni ed usati motori di ricerca, come ad esempio Google.

Sono presenti delle liste di *stopwords* predefinite, nelle diverse lingue, in molti pacchetti e librerie di software di programmazione. In questa applicazione è stata adoperata la **library (tm)** di Rstudio che utilizza una lista di *stopwords* in italiano provenienti dallo "Snowball stemmer project" (`https://snowball.tartarus.org/`).
Quest'ultimo è un portale storico che aveva come obiettivo la realizzazione di algoritmi di *stemming* e *stoplist* in varie lingue, che sono tutt'ora a disposizione del pubblico.

Inoltre è possibile creare un insieme di parole personalizzate da poter

adoperare come *stopwords*.

```
#remove personalized stop words from the row

myStopwords<-c("etc","ecc","xxx","NA","esemp","eo","esegu","esse","altre","mett","es","no"
,"altri",",altro","tutt","poss", "labor","cod","lg","mess","eserciz","x","not","d","bc"
,"modif","mix","mov", "ved","abc","abb","abitual","alt","alta","inclus","utilizz","relat"
,"attiv","ben","includ","tip","gener","azi","continu","esclus","oem","prim","grand","bas"
,"classific","rif","prov","senz","sott","assoc","effetu","indirett","permanent","pian"
,"tratt","xx","alcun","ambit","altres","are","az","bass","comau","complet","duc","vien","vw"
,"qual","tradizional","succ","sol","var","similar","facent","up","ogn","offron","esser"
,"inoltr","appartenent","apparteng","bert","divent","fin","og","hilt","effetu","già","part"
,"lattiv","srl","appart","dat","pian","part","gest","general","mediant")
```

Figura 4.2: *Insieme di stopwords personalizzate*

In Fig. 4.2 sono presenti alcuni termini, che sono stati adoperati dopo lo *stemming* (tecnica che verrà presentata nel paragrafo successivo), per essere filtrati dal documento "Normativa TRADE 2022.xls". In tal caso le *stopwords* sono risultate utili nel migliorare la *performance* del classificatore basato sulla *Cosine Distance*. Infatti, il documento in questione presentava poche pagine ed alcuni termini che sarebbero stati valutati irrilevanti in testi più lunghi non venivano considerati come tali. Ad esempio parole come "altrove","relativo" etc. sarebbero potute risultare importarti nel calcolo della similarità tra documenti.
Pertanto, si è deciso di ometterli dal documento in questione tramite una *stoplist* personalizzata.

4.1.3 Stemming

Lo *Stemming* è il processo di riduzione dei termini alla loro radice, detta anche tema. Infatti, alcune parole possono presentare diverse forme e declinazioni che riconducono ad un medesimo significato. Ad esempio le parole gatto, gatta, gattino, gattile etc. possono essere associate alla radice "gatt". Il tema non ha necessariamente un significato esplicito e rappresenta solamente la troncatura dei vari termini.
Anche in questo caso lo *stemmer* adoperato proviene dallo "Snowball Stemmer Project". Il metodo in questione è stato ideato da Porter nel

1980 [29] per la lingua inglese, successivamente mediante il progetto sopracitato è stato ampliato a diverse lingue, pertanto è riconosciuto anche con il nome di "Porter2". Tutt'ora risulta il metodo più largamente utilizzato grazie alla sua precisione e velocità computazionale.
Tuttavia le tecniche di *stemming* non tengono in considerazione che due termini potrebbero avere lo stesso significato, ma radice diversa.
Ad esempio, le parole "abitazione", "edificio", "casa" e "palazzo" hanno il medesimo significato, ma un algoritmo di *stemming* non riesce a comprendere questi sinonimi poiché presentano radici diverse.

Per porre rimedio a questo problema è stata introdotta un ulteriore metodologia che prende il nome di ***Lemmatization***, consiste nella riduzione delle parole alla loro radice morfologica, detta *lemma*.
Si distingue dallo *stemming* in quanto in questo caso l'interesse è quello di accorpare il più elevato numero di termini che presentano lo stesso significato, in questo modo si riesce a ridurre la complessità del testo.
Ad esempio, il lemma delle forme verbali (sono, sei, è, siamo, siete, sono) riconduce al verbo all'infinito (essere). Per riuscire in questo intento, solitamente si utilizzano enormi database semantico-lessicali, come ad esempio WordNet. Pertanto, il suo utilizzo risulta computazionalmente molto pesante, sopratutto se si considerano dataset di grandi dimensioni. Per questo motivo si è deciso di non adottarla in questa applicazione.

4.2 Bag Of Words

Successivamente è fondamentale rappresentare le parole in maniera tale che possano essere inserite come *input* nei modelli di *Machine Learning*. Per fare ciò si convertono le frasi in vettori numerici, procedimento che prende il nome di *text vectorization*.

> *"In language processing, the vectors x are derived from textual data, in order to reflect linguistic properties of the text."* [30]

Dato che in questo caso si hanno a disposizione un insieme di n documenti $\mathbf{D} = \{d_1, \ldots, d_n\}$, si otterà una *Document Term Matrix* (DTM), ossia una matrice che rappresenta il testo in formato numerico.
Nella Fig. 4.3 si visualizza un esempio di DTM, che è stata costruita mediante il dataset presentato precedentemente.

	costruzion	edif	residenzial	costruzion edif	edif residenzial	install	impiant	idraul	riscald
text1	1	1	1	1	1	0	0	0	0
text2	1	1	1	1	1	0	0	0	0
text3	0	0	0	0	0	1	1	1	1
text4	0	0	0	0	0	0	0	0	0
text5	1	1	0	0	0	1	1	1	1

Figura 4.3: *Esempio di Document Term Matrix (booleana)*

La DTM è un implementazione del concetto noto come *Bag Of Words* (BoW). Quest'ultima tecnica consiste nel considerare i *token*, ottenuti dai passi precedentemente descritti, ed inserirli in una "bag" (in italiano borsa) indipendentemente dall'ordine delle parole, considerando solamente dei termini di frequenza e perdendo l'informazione sulla posizione delle parole.
La matrice numerica che si ottiene è composta da:

- **Document**: termine generico che indica un documento testuale, in questo caso tutti gli oggetti sociali sono racchiusi in un unico file, quindi vengono considerati come blocchi di testo ($text_i$), dove i sta ad indicare il generico documento ;

- **Term**: indica il *token* che abbiamo scelto di considerare. Pertanto nel nostro caso saranno compresi termini di parole singole e coppie di parole consecutive (bigrammi).

Il procedimento che assegna un peso specifico ad ogni termine del testo prende il nome di ponderazione del testo, oppure in inglese *Term Weighting Scheme* (TWS). Una delle prime proposte per pesare i termini, è stata fatta da H.P. Luhn nel 1957 [31], il quale propose un metodo automatico per processare i testi basato sulla frequenza delle parole.
Gli schemi di ponderazione più conosciuti ed adoperati sono [32]:

1. il **booleano**, in cui w_{ij} assume valore 1 se la parola i è presente nel documento j e 0 altrimenti;

2. il **frequentista**, in cui w_ij è uguale a n_{ij}, frequenza della parola i nel documento j;

3. il **frequentista normalizzato**, in cui $w_{ij} = n_{ij}/ \max n_j$, con max n_j frequenza della parola più presente nel del documento j

4. **tf/idf** (term frequency / inverse document frequency), consente di valutare il peso di un termine all'interno di un documento:

$$w_{ij} = tf_{ij}.log(\frac{N}{df_i})$$

   Dove le espressioni della formula rappresentano:

   - tf_{ij} = numero di volte che il termine (i) si presenta nel documento (j), tale metrica può anche venir normalizzata;
   - df_i = numero di documenti contenenti il termine (i)
   - N = numero totale di documenti (j)

   Quest'ultimo metodo è stato proposto da Salton nel 1989 [33] come algoritmo per migliorare la valutazione dei termini nell'ambito dell'IR (Information Retrieval).

L'introduzione dell'IDF (*Inverse Document Frequency*) vuole penalizzare le parole che appaiono in un numero elevato di documenti simili. Pertanto questa metrica serve per dare alle parole definite precedentemente come *stopwords* valori molto bassi, in modo tale che vengano considerate poco nella classificazione. Pertanto, quando viene già adottata una *stoplist* i benefici che si avranno dall'utilizzo di questa tecnica non saranno molto apprezzabili, anche se potrebbe comunque filtrare alcune parole poco significative che non erano state incluse nelle *stopwords*

Per costruire la DTM mi sono servito del pacchetto **library (quanteda)** ed ho implementato un'ulteriore *pipeline* mediante il codice rappresentato in Fig. 4.3. Inizialmente sono state accorpate in un unico oggetto le descrizioni "pulite" tramite la funzione di *stemming* descritta in precedenza.
Successivamente, sono state filtrate le *stopwords* personalizzate ed infine mi sono servito della libreria quanteda per suddividere il testo in *tokens* e calcolare la ponderazione del testo. Nel caso in Fig. 4.3 si è adoperata la tecnica **tf/idf**, ma esistono altre numerose opzioni che si possono considerare, tra cui quelle descritte negli schemi di ponderazione.

```r
library(quanteda)
create_dtmIDF = function(data){
  set_spec  <- stemm(data$des_ateco)
  set_desc  <- stemm(data$oggetto_sociale)

  trade_tot<-c()

  for( i in 1:length(data$cod_cliente)){
    if (set_desc[i]!="NA"){
      trade_tot[i]=paste(set_spec[i])
      trade_tot[i]=str_split(trade_tot[i],"[^[:alnum:]]+")
      trade_tot[[i]]=unique(trade_tot[[i]])

    }else{
      trade_tot[i]=paste(set_spec[i],set_desc[i])
      trade_tot[i]=str_split(trade_tot[i],"[^[:alnum:]]+")
      trade_tot[[i]]=unique(trade_tot[[i]])
    }
  }

  for(i in 1:length(trade_tot)){
    trade_tot[[i]]=trade_tot[[i]][!trade_tot[[i]] %in% myStopwords]
  }

  dtm <- quanteda::dfm(tokens_ngrams(tokens(trade_tot),n= 1:2, concatenator = " "))
  #trimming the most frequents term
  dtm_trim <-
    quanteda::dfm_trim(
      dtm,
      min_docfreq = 0.01,
      docfreq_type = "prop")

  #arg_tf:  "count", "prop", "propmax", "logcount", "boolean", "augmented", "logave"
  #arg_scheme = c("count", "inverse", "inversemax", "inverseprob", "unary")
  dtm_idf <- quanteda::dfm_tfidf(dtm_trim,
                                 scheme_tf = "count",
                                 scheme_df = "inverse",
                                 base= 10,
                                 force = F)
  dtm_matrix <-as.matrix(dtm_idf)
  return(dtm_matrix)
}

```

Figura 4.4: *Pipeline per la creazione e personalizzazione della DTM*

Capitolo 5

Applicazione dei Modelli

Ora che il testo è stato trasformato in un formato numerico e quindi interpretabile dai classificatori, si può proseguire alla loro applicazione. Nel caso della *Text Classification* saranno proprio le parole (singole e bigrammi) le variabili dei modelli ed il loro valore sarà relativo al metodo di ponderazione del testo adottato nella DTM.

I modelli di ML possono essere suddivisi in:

- ***Supervised Learning*** principalmente applicato in compiti relativi alla classificazione o regressione;
- ***Unsupervised Learning*** generalmente adottati per effettuare *clustering*, associazioni tra variabili e riduzione della dimensionalità.

La sostanziale differenza tra i due metodi è che il *Supervised Learning* viene allenato su dei dati già etichettati. Per questa applicazione i dati dei clienti erano stati precedentemente catalogati dai dipendenti dell'azienda nel corso degli anni e sono stati in parte etichettati da me durante il tirocinio.

Il dataset è stato suddiviso in una parte di *training* (partizione dell' 80%) ed in una di *testing* (rimanente 20%).
Il classificatore dopo essere stato adeguatamente allenato sui dati di *training* viene testato sui dati restanti privati dell'etichetta, che viene adoperata in fase successiva per valutare la bontà della classificazione ottenuta.
In realtà anche nel caso dell'*Unsupervised Learning* in questo caso avremo delle categorie già predefinite per testare l'*accuracy* del raggruppamento dei documenti tramite la distanza *cosine*, però non sarà presente l'allenamento dei modelli in quanto è un metodo non supervisionato.
Come si potrà notare successivamente nel Capitolo 6 le tecniche supervisionate ottengono una migliore $performance$ in termini di classificazione.

Inoltre, prima di procedere all'applicazione dei modelli in molti casi reali è necessario trasformare la variabili. Infatti, in molti casi possono esserci delle incongruenze tra il *testing* ed il *training* set o limiti di variabili categoriche.
In questo capitolo verranno introdotti sia i classificatori adottati sia le tecniche ed i metodi per standardizzare i dati e migliorare le $performance$.

5.1 Unsupervised Learning

5.1.1 Cosine Similarity

La similarità del coseno è una tecnica non supervisionata per la misurazione della similitudine tra due vettori. Viene misurata calcolando il coseno tra i due vettori per valutare quali tra questi siano più "vicini".
Anche se non è considerata una vera e propria metrica in quanto non rispetta la disuguaglianza triangolare. Il suo valore è compreso tra [0,1].

Considerati due vettori A e B, la loro similarità del coseno è formalmente:

$$similarity(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \, \|B\|}$$

Come possiamo vedere in Fig. 5.1, i vettori che stanno puntando approssimativamente nella stessa direzione sono considerati *simili*.

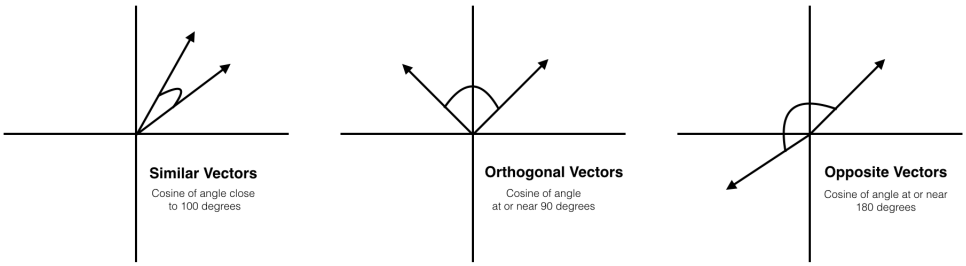

Figura 5.1: *Grafici rappresentanti la distanza coseno tra due vettori*

Questa distanza viene spesso adoperata per calcolare la similarità tra documenti nell'ambito della *text analysis*. Infatti, le metodologie che sono state descritte nel capitolo precedente potranno essere utilizzate per trasformare il testo in *text vectors*.

Per l'applicazione della similarità del coseno sono stati considerati i seguenti testi:

1. descrizioni aziendali delle 5 attività economiche considerate in questa applicazione, presentate nel Paragrafo 3.5, e provenienti dal file "Normativa TRADE 2022.xls";

2. L'oggetto sociale e la descrizione sintentica del codice ateco, provenienti dalla piattaforma Atoka Cerved.

Queste descrizioni sono state rispettivamente trasformate in vettori testuali, in modo tale da poterle confrontare tra loro. Il vettore testuale ottenuto dal secondo punto è stato confrontato con ognuno dei 5 vettori testuali ottenuti dalle descrizioni aziendali.

Le distanze ottenute sono state inserite in un vettore e quella più elevata è servita da discriminante per la classificazione. Alcune di queste distanze presentavano il valore 0, questo avviene quando si hanno due vettori che non presentano alcun elemento in comune. In questo caso, si è scelto arbitrariamente di attribuire la categoria "altri servizi commerciali" (42200), in quanto quest'ultima aveva una descrizione formale dell'azienda molto indicativa. Inoltre, rappresenta un codice che al suo interno racchiude molteplici attività economiche.

Come metodo di ponderazione del testo è stata usata la frequenza, ossia sono state conteggiati i *token* presenti nei rispettivi vettori testuali.
Il pacchetto adoperato è stato **library (lsa)**, il cui acronimo sta ad indicare *Latent Semantic Analysis*, tale libreria ha al suo interno una funzione in grado di calcolare la distanza angolare *cosine* tra documenti.

In Fig. 5.2 si può osservare il codice adoperato per calcolare la distanza del coseno.

```
#COSINE DISTANCE
library(lsa)
cosine_distance = function(descrizione){
  des<-stemm(descrizione)
  des<-str_split(des,"[^[:alnum:]]+")
  trade_tot[6]<-des
  dtm <- dfm(tokens_ngrams(tokens(trade_tot),n= 1:2, concatenator = " "))
  score_vector<-c()
  for (i in 1:length(trade_tot)){
    score_vector[i]<-lsa::cosine(as.vector(dtm[i,]),as.vector(dtm[6,]))
  }
  if((score_vector[which.max(score_vector[-6])])==0){
    return(5)
    }else{
      return(which.max(score_vector[-6]))
  }
}
```

Figura 5.2: *Codice per calcolare la distanza del coseno*

La *Cosine Similarity* ha come vantaggio quello di essere un metodo *unsupervised* di classificazione, ovvero non necessità di un dataset di *training*. Pertanto, può essere adoperato euristicamente nel caso di dataset poco forniti o come metodo di classificazione per documenti e dati

che non sono stati categorizzati. Inoltre, può essere adoperato come variabile in altri modelli, in quanto ci restituisce una misura approssimativa della vicinanza tra due vettori.

5.2 Supervised Learning

5.2.1 Naive Bayes Classifier

Il classificatore *Naive* Bayes è un algoritmo probabilistico basato sul teorema di Bayes. Il classificatore in questione viene detto "naive" poiché presuppone che tutte le variabili predittive siano indipendenti tra loro. È un algoritmo *supervised* molto efficace e rapido, ossia non richiede molta potenza di calcolo, per questo motivo è stato usato fin da subito nelle implementazioni di ML [15].
Una delle sue più note applicazioni è stata per il filtraggio delle email *spam* a fine anni Novanta.

Il suo calcolo si basa sul calcolo delle probabilità dati due eventi A, B:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

- $P(A \mid B)$: probabilità che l'ipotesi A si realizzi data l'osservazione dell'evento B (probabilità a posteriori);
- $P(B \mid A)$: probabilità che data l'evidenza B l'ipotesi A sia vera (probabilità di verosimiglianza);
- $P(A)$: probabilità che l'ipotesi si realizzi, prima di osservare le evidenze (probabilità a priori);
- $P(B)$: probabilità dell'evidenza B (probabilità marginale).

Il classificatore è stato implementato attraverso il pacchetto "e1071" e la sua apposita **library (naiveBayes)**.
Può succedere che un certo *token* o codice ateco non vengano considerati

nei dati di *training*, in questo caso risulta utile la correzione di Laplace, che consiste nell'aggiunta di un conteggio per tutte le variabili. Questo *smoothing* permette di poter adoperare questo classificatore anche senza procedere alla *Feature Engineering*, che verrà trattata nel Paragrafo 5.3.

5.2.2 Support Vector Machines (SVM)

I classificatori a vettori di supporto (SVM) sono stati sviluppati nel 1965 da Vladimir Vapnik, un matematico sovietico [16]. Inizialmente, erano stati sviluppati per risolvere problemi di classificazione binaria, ma sono stati poi ampliati anche per la classificazione multiclasse.
È un classificatore *supervised* che riesce a mappare bene vettori multidimensionali e, pertanto, ben si presta in compiti di *Text Classification*. E stato adottato per la prima volta con successo in compiti di classificazione del testo nel 1998 da Joachims [34].

I classificatori SVM si suddividono in:

- **SVM lineare**, la superficie di separazione è un iperpiano;
- **SVM non lineare**, utilizzano dei *kernel trick*, ossia delle funzioni che determinano la forma ed i confini dell'iperpiano, che in questo caso sarà non lineare.

Verrà introdotto maggiormente nel dettaglio il concetto alla base dei **SVM lineari**, in quanto sono il metodo di classificazione applicato in questo progetto.
L'obiettivo dei classificatori SVM lineari è quello di trovare un iperpiano in uno spazio n-dimensionale (n sono il numero di variabile esplicative) che minimizza il numero di elementi non correttamente classificati. Inoltre, allo stesso tempo deve andare a massimizzare il *margine* tra le classi, dove con *margine* si intende la distanza dell'iperpiano dai punti più vicini delle diverse classi. I *support vectors* sono proprio quei vettori che sono più vicini all'iperpiano e sono fondamentali per la costruzione

del modello di classificazione.

Anche in tal caso è stata adoperato il pacchetto **library (e1071)**, che contiene al suo interno una libreria specifica per i classificatori SVM.
In Fig. 5.3 si può visualizzare il codice usato per implementare il modello.

```
library(e1071)
mod_svm <-svm(as.factor(trade)~.,data=dtm_train_matrix,
              type = "C-classification",

              #kernel: "linear","polynomial","radial basis","sigmoid"
              kernel = "linear",

              #10-fold Cross Validation
              trControl=trainControl(method='cv',number=10))
```

Figura 5.3: *Codice per implementare classificatore SVM*

Inoltre, è stata adoperata la k-fold ***cross validation***, la quale consiste nell'ulteriore suddivisione dei dati di training in un numero di $fold$ (in questo caso k=10) per validare il modello di ML ed evitare situazioni di $overfitting$. Infatti, il processo di addestramento e di valutazione viene ripetuto k volte, utilizzando ogni volta una partizione diversa come set di valutazione ed addestrando il modello sui dati rimanenti. Alla fine, vengono calcolate le medie della precisione dei modelli valutati in modo da ottenere una stima più accurata della precisione sul test set.

5.2.3 Random Forest

I modelli *Random Forest* sono dei metodi di apprendimento supervisionati e non parametrici che possono essere adoperati in compiti sia di regressione che classificazione. Sono stati introdotti solamente nel 2001 da Breiman [17] e proprio in questo periodo ci sono state le prime applicazioni pratiche di successo della *Text Classification* (come ad esempio il filtering della mail), per cui questo metodo è stato fin da subito adottato con ottimi risultati, pertanto risulta interessante testarli in quest'ambito. Come può suggerire il nome rappresentano un aggregato di alberi decisio-

nali, che sono a loro volta un algoritmo di apprendimento, ma i Random Forest hanno ottenuto una migliore *performance* grazie alla combinazione di multipli alberi casuali.

Fanno parte di una tecnica di apprendimento di insieme (***ensemble***, in inglese). In particolare, la tipologia di insieme di questo algoritmo è il *bagging*, ossia più modelli dello stesso tipo vengono addestrati su insiemi di dati diversi ottenuti tramite campionamenti casuali con reinserimento (gli alberi decisionali sono tra loro indipendenti). Questo tipo di struttura li rende particolarmente solidi sia riguardo a problemi di dati mancanti che di *overfitting*, infatti non risulta necessario procedere con una *cross validation*, come abbiamo invece fatto per gli altri due modelli supervisionati.
Il concetto alla base di questo metodo è la teoria che "più teste sono meglio di una". Infatti, l'obiettivo è quello di creare tanti classificatori "deboli" che insieme possano dare un buon risultato.

In tal caso è stata adoperata la **library (randomForest)** di R che viene appositamente utilizzata per il training ed ha alcuni parametri che possano essere adoperati per regolare il comportamento del modello. In particolare abbiamo:

- **ntree**: specifica il numero di alberi da costruire nella random forest, più il numero è elevato più sarà la precisione, purtroppo anche la complessità ed il tempo di *training* dell'algoritmo aumenteranno. Se nessun valore viene specificato viene automaticamente settato a 500 alberi decisionali;

- **mtry**: specifica il numero di variabili da considerare in ogni split di ogni albero, se non specificato corrisponde al radice quadrata del numero di colonne y del dataset;

- **importance**: è stato impostato su TRUE, pertanto il modello calcolerà l'importanza delle variabili (di default è FALSE).

Inoltre i modelli *Random Forest* sono in grado di darci una semplice e grafica interpetazione dei risultati ottenuti. Ad esempio, in tal caso avevo chiesto al modello di considerare l'importanza delle variabili nei diversi nodi decisionali. Pertanto, possiamo visualizzare il grafico che poi è stato ottenuto mediante la funzione "varImpPlot" in Fig. 5.4.

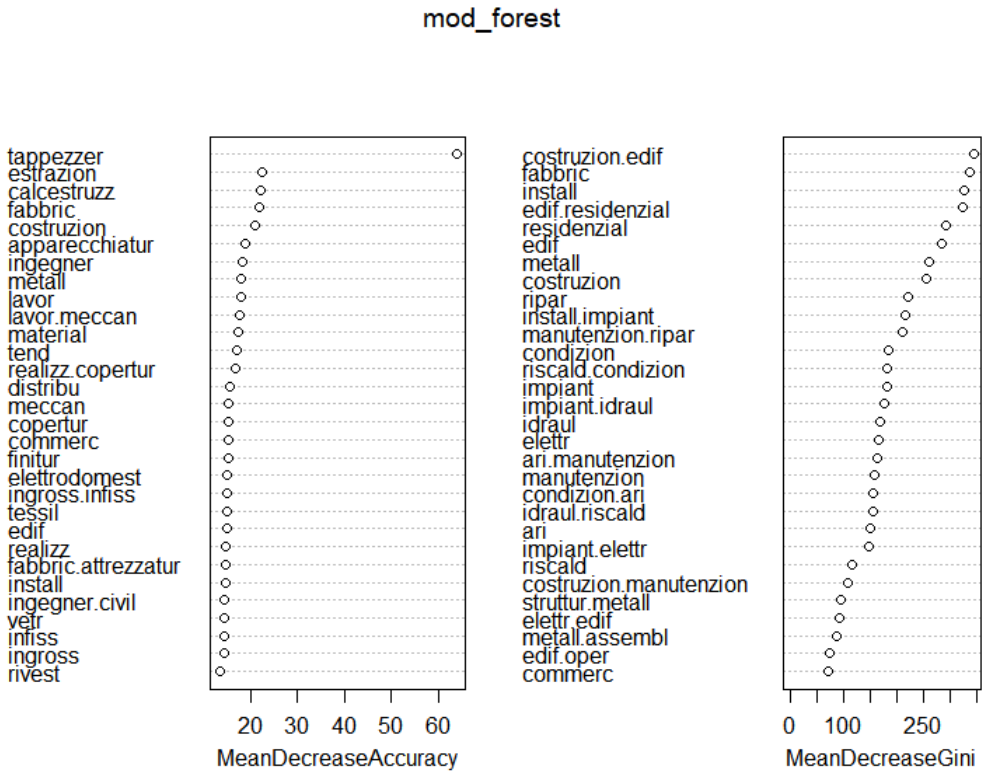

Figura 5.4: *Variabili risultate importanti per modello RF*

5.3 Feature engineering

La *Feature engineering* si riferisce ad un processo di trasformazione delle variabili grezze in caratteristiche più significative, in modo tale da migliorare i risultati dei modelli di ML.
Da non confondere con la selezione delle variabili che viene adottata in fase di *pre-processing* dei dati. In realtà, anche tecniche di *Feature engineering* possono essere adoperate in fase di preparazione dati, come ad esempio per l'imputazione di dati mancanti. In questo caso però sono servite per migliorare la performance dei modelli e, pertanto, verranno presentate in questo capitolo.

Alcune metodologie che vengono frequentemente adottate sono:

- **Riduzione delle caratteristiche**, consiste nella selezione degli input del modello. La riduzione degli attributi può essere svolta attraverso:
  - Analisi esplorativa dei dati (EDA, dall'inglese *Exploratory Data Analysis*);
  - *Feature Extraction*: consiste nella derivazione ed estrazione di nuovi attributi da quelli esistenti;
  - *Filter-based*: gli attributi irrilevanti vengono filtrati;
- **Trasformazione delle variabili**, consiste nell'utilizzo di funzioni matematiche o statistiche per trasformare i dati. Può servire per aggiustare delle distribuzioni asimmetriche. Alcuni metodi adottati sono ad esempio la normalizzazione o standardizzazione;
- **Creazione di nuove variabili**, consiste nella generazione di nuove variabili a partire da quelle esistenti;
- ***Encoding categorical data***, consiste nel trasformare in valori numerici le variabili categoriche. Un esempio è il *one-hot enco-*

ding che viene adoperato quando abbiamo una variabile categorica binaria, ad esempio se il paziente fuma o meno (1,0).

L'utilizzo della *Feature engineering* è risultato fondamentale soprattutto per implementare il classificatore a vettori di supporto (SVM). Poichè, quest'ultimo modello non è basato su un concetto probabilistico ma geometrico. Pertanto, situazioni nelle quali vi erano alcune differenze fra gli attributi presenti nel *training* e nel *test* set portavano al non funzionamento del modello. Inoltre, non era possibile adoperare soluzioni come il *Laplace Smoothing*, tecnica adottata per il classificatore di Bayes nel Paragrafo 5.2.1, che consiste in una piccolo incremento della probabilità per ogni attributo.
Situazioni in cui *token* ed alcuni codici ateco (presentanti nel paragrafo 3.2) non vengano considerati nei dati di *training*, o rispettivamente nel *test set*, possono accadere quando:

1. **codici ateco** che sono piuttosto rari o datati (considerano una vecchia codificazione);

2. **token** incongruenti, ossia quando alcune variabili testuali, non si presentano rispettivamente sia nel *training* che nel *test* set. Ad esempio, può accadere che il token "case prefabbricate" si presenti un paio di volte all'interno dei dati di allenamento (che sono anche più numerosi), ma nemmeno una volta nei dati di *testing*.

Per porre rimedio a questi problemi sono state adottate alcune tecniche di *Feature engineering*, in particolar modo di riduzione della dimensionalità per livellare le due partizioni di dati.

Per quanto riguarda i codici ateco, sono state considerate solamente le prime 4 cifre per evitare che ci fossero codici non riscontrati, pertanto è stata adottata una tecnica di *Feature extraction* che va a ridurre la complessità del modello. In pochi casi, sono comunque rimasti dei codici ateco che non venivano riscontrati in entrambe le partizioni, pertanto è

stata creata un ulteriore variabile "other" da assegnare a questi codici ateco per uniformare i dati. Questi passaggi sono stati svolti tramite una funzione, che adotta delle semplici condizioni logiche, riportata in Fig. 5.5.

Per adoperare i codici ateco nel modello *Random Forest* si è dovuta aggiustare la funzione presentata in Fig. 5.5 per la standardizzazione dei codici ateco. In quanto la libreria "randomForest" di Rstudio non accettava più di 53 classi (o livelli) nelle variabili categoriche, pertanto sono stati considerati solamente i primi 2 numeri della codificazione ateco.

Per quanto riguarda i *token*, invece è stato adottato una tecnica *filter-based* per filtrare quelle parole/frasi che non si presentavano in entrambi i dataset. Da considerare che solamente gli attributi in questione sono stati omessi dalle DTM e non sono stati eliminati dei dati.

```
#standardization function for the ateco_codes
standard_ateco<-function(dtm_matrix,dtm_matrix2){

  #truncating the ateco code (4char)
  for(i in 1:length(dtm_matrix$cod_ateco)){
    if(nchar(as.numeric(dtm_matrix$cod_ateco[i]))>3){
      dtm_matrix$cod_ateco=substr(dtm_matrix$cod_ateco,1,4)
    }else{
      dtm_matrix$cod_ateco=dtm_matrix$cod_ateco
    }
  }
  as.factor(dtm_matrix$cod_ateco)

  #the ones that differs in the 2 matrix
  #are replaced with "other"
  diff<-dtm_matrix$cod_ateco%in%dtm_matrix2$cod_ateco

  for(i in 1:length(diff)){
    if (diff[i]==FALSE){
      dtm_matrix$cod_ateco[i]="other"
    }
  }
  return(dtm_matrix)
}
```

Figura 5.5: *Funzione per livellare e standardizzare i codici ateco*

5.4 Hyperparameter Tuning

Il tuning degli iperparametri è l'ambito che si occupa della scelta dei parametri ottimali per un modello di ML. Esistono dei parametri già preimpostati per ogni libreria, ma può essere utile sperimentare altri insiemi di valori per cercare di ottenere una migliore performance.
Infatti, gli iperparametri si discostano dai parametri del modello, in quanto non vengono ottenuti da un processo di apprendimento, ma vengono scelti a priori e reiterati per trovare la miglior combinazione. Per questo motivo è un procedimento che richiede molto tempo di esecuzione, anche in base alla quantità dei parametri che si decide di testare.

Di seguito verranno presentati 3 metodi principali per ottimizzare gli iperparametri:

- ***Grid Search***: un metodo esaustivo di ricerca tramite un insieme di liste manualmente preimpostate su alcuni valori parametrici. Di fatto vengono esplorate tutte le combinazioni possibili dei valori presenti nelle liste. Per questo motivo è uno dei metodi computazionalmente più pesanti, ma generalmente garantisce ottimi risultati;

- ***Random Search***: è una semplice alternativa al metodo descritto precedentemente. In tal caso viene esplorato solo un sottoinsieme estratto casualmente della *grid*, in questo modo si alleggerisce anche computazionalmente il processo di ottimizzazione. Tuttavia c'è sempre la possibilità di non riuscire ad individuare una configurazione ottimale dei parametri;

- ***Informed Search***: in questo caso ogni iterazione impara dalla precedente. In quanto i risultati ottenuti dal modello precedente ci guidano nella ricerca successiva degli iperparametri ottimali. Il più popolare metodo di ricerca informata è la *Bayesan Optimization*, originalmente progettata per ottimizzare le funzioni *black-box*.

Per quanto riguarda questo progetto sono state costruite delle *gridSearch* da poter utilizzare con i modelli a vettori di supporto e *Random Forest*. La libreria utilizzata è stata *tune*, sempre del pacchetto "e1071", che presentava la possibilità di allenare questi modelli. Il classificatore Naive Bayes non era presente nelle modalità di scelta e data la sua semplicità generalmente non necessità di essere ottimizzato.
Ogni modello presenta dei diversi parametri che possono essere ottimizzati attraverso i metodi di *hyperparameter tuning*, questi però differiscono in base all'algoritmo considerato. Ad esempio, il modello SVM presenta come parametri da ottimizzare:

- **gamma**: rappresenta le linee di confine che cercano di raggruppare insiemi di punti con caratteristiche simili, più è piccola più vengono considerati degli "spazi" ampi per suddividere i dati;

- **C**: rappresenta il costo-penalità che viene assegnato in caso di classificazione errata. Quando viene considerato molto alto, si considera un *margine* che suddivide i punti delle diverse classi molto sottile, in tal caso prende il nome di *hard* SVM. Invece, quando viene considerata un valore di C piuttosto basso abbiamo un margine più ampio per la $misclassification$ e si considera un $soft$ SVM.

Per quanto riguarda questa applicazione, si sono ottenuti i migliori risultati con valore di gamma pari a 0.0012 ed un costo C di 1, pertanto verrà considera un margine più ampio e relativamente si tratta di un $soft$ SVM.

Invece, per quanto riguarda il classificatore *Random Forest* si considerano come parametri da ottimizzare il numero di alberi (**ntree**) ed il numero di variabili da considerare in ogni ramo (**mtry**). La presentazione di quest'ultimi valori è presente al paragrafo 5.2.3.

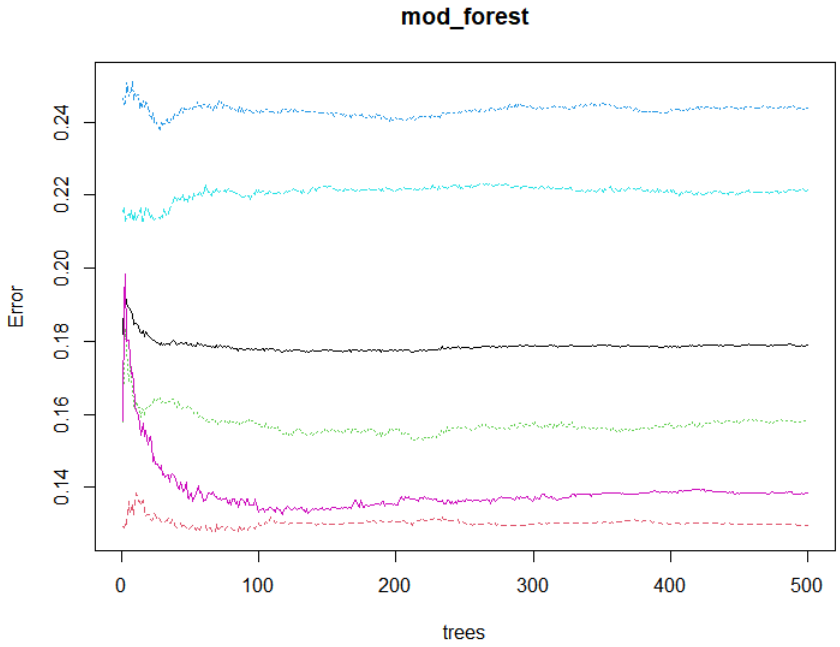

Figura 5.6: *Grafico rappresentante accuracy relativa al numero di alberi considerati*

Ad esempio, se si considerano un numero maggiore di alberi (ovviamente sempre riferito al modello di *Random Forest*), si avrà una miglior *accuracy* generale. Il risultato di questo esperimento può essere visibile in Fig. 5.6. Bisogna anche considerare che l'*error rate* risulta leggermente inferiore a quello che si otterrà nel *test* set. In quanto il grafico è relativo ai dati raccolti durante l'allenamento, che tende a sovrastimare leggermente la $performance$.

Capitolo 6

Risultati degli esperimenti

In questa sezione verranno raccolti e presentati i risultati della classificazione ottenuti dai modelli di ML.
Lo svolgimento degli esperimenti è stato svolto su un dataset composto da un campione casuale di n=20000 documenti, che sono stati ricavati dalle osservazioni descritte nel Par. 3.5.
La scelta di ridurre la numerosità del dataset è stata fatta poiché si avevano molte informazioni a disposizione e considerare tutti i dati andava solamente ad aumentare la complessità computazionale dei modelli. Infatti, oltre alle metriche relative alla classificazione verrà considerato anche il tempo di esecuzione, in minuti, per ogni algoritmo.
Per la scelta dei parametri sono state considerati i termini che si ottengono con gli unigrammi k = 257 *tokens* e rispettivamente con i bigrammi k = 545 *tokens* (riferimento al Paragrafo 4.1.1). Il testo da cui sono state ricavate le variabili sono la descrizione sintetica del codice ateco e l'oggetto sociale. Successivamente è stato aggiunto il codice ateco standardizzato per confrontare i risultati.

6.1 Metriche

Un metodo molto adottato per valutare per $performance$ di modelli di classificazione è la matrice di confusione (in inglese conosciuta come

confusion matrix). Quest'ultima è una matrice CxC (dove C sta ad indicare il numero di classi), che mostra la relazione tra le etichette previste e quelle reali. Viene mostrato un esempio di matrice di confusione in Fig. 6.1.

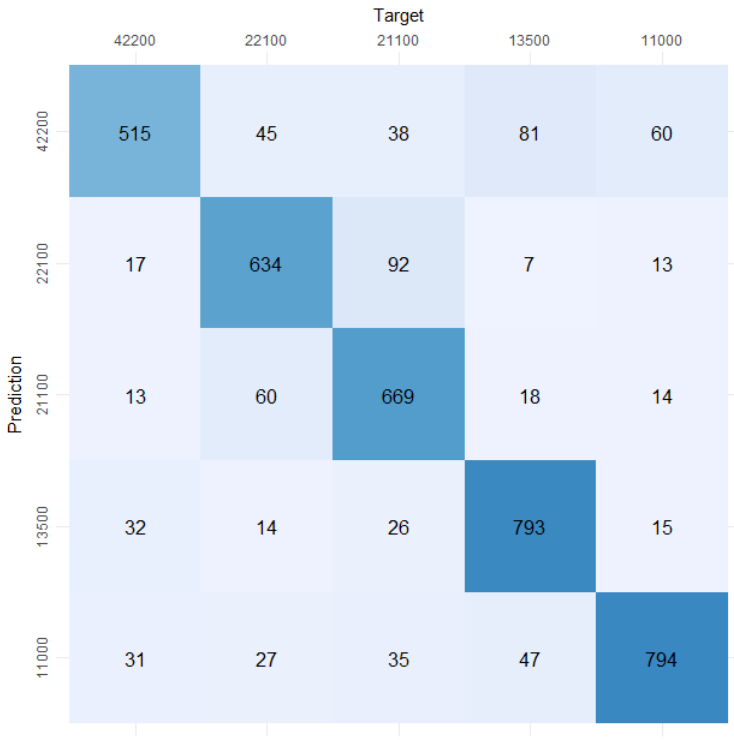

Figura 6.1: *Matrice di confusione del modello SVM-2w con codice ateco*

Lungo la diagonale sono riportati i valori che sono stati correttamente classificati (FNi), mentre a destra della diagonale sono presenti i falsi negativi (FNi) e rispettivamente a sinistra i falsi positivi (FPi) per ogni classe $i = 1, ..., C$.

La matrice di confusione è uno strumento utile per calcolare l'accuratezza, precisione, *recall* ed *F1-score* del modello.

L'accuratezza è la metrica che definisce la *ratio* tra il numero di classificazioni correttamente classificate e il totale delle osservazioni (N), pertanto è una misura globale di $performance$.

$$Accuracy = \frac{|i : \hat{y}_i = y_i, \forall i = 1, ..., C|}{N}$$

La precisione, il *recall* e l'*F1-score* vengono invece calcolate in riferimento ad ogni classe.

Inizialmente, è stata calcolata l'*accuracy* sul *test-set* (composto da 4090 osservazioni) dei diversi algoritmi considerando solamente i dati testuali rispettivamente con 1-gram (1w) e 2-gram (2w). Si precisa che il tempo misurato è unicamente relativo alla durata del *training* dell'algoritmo, compresa la k-fold *Cross Validation*, e non considera il tempo per la creazione della DTM (*Document Term Matrix*), la preparazione delle variabili ed il *tuning* degli iperparametri.
Presentazione dei risultati in Fig. 6.2.

model (no cod)	CD - 1w	CD - 2w	BAY - 1w	BAY - 2w	SVM -1w	SVM -2w	RF - 1w	RF - 2w
accuracy	**63.35%**	**72.4%**	**80.03%**	**80.042%**	**82.54%**	**82.9%**	**82.71%**	**82.98%**
time (min)	6.92	6.92	0.317	0.687	1.67	2.95	11	11.47

Figura 6.2: *Tabella che raffigura l'accuratezza globale dei diversi modelli*

Si può notare come la considerazione dei bigrammi è stata di particolare rilevanza solamente nell'algoritmo della *Cosine Distance* (CD), in tutti gli altri modelli supervisionati l'aumento dell'accuratezza è molto meno evidente. Inoltre, pare fin da subito evidente che i modelli SVM ed i *Random Forest* (RF) hanno una $performance$ nettamente maggiore rispetto agli altri algoritmi.

Successivamente, sono stati considerati solamente i modelli supervisionati ed è stato aggiunto il codice ateco ai *tokens*. Inoltre, sono stati considerati anche i risultati dell'*accuracy* relativi al *train-set*, infatti è bene confrontare anche tali risultati per valutare la presenza di $overfitting$.
Da come si può notare in Fig. 6.3 i risultati sul *train-set* sono praticamente equivalenti e solo in alcuni casi leggermente superiori.

Purtroppo, l'aggiunta del codice ateco ha migliorato solo leggermente i risultati ed anche i risultati a confronto tra unigrammi e bigrammi si equivalgono.

	Test - set					
model (cod)	BAY - 1w	BAY - 2w	SVM -1w	SVM -2w	RF - 1w	RF - 2w
accuracy	**81.3%**	**81%**	**83.33%**	**83.25%**	**82.8%**	**83.2%**
time (min)	0.3	0.62	1.22	2.98	6.1	10.6
	Train - set					
accuracy	**81.2%**	**80.85%**	**84.6%**	**84.7%**	**82.91%**	**83.4%**
time (min)	1.23	2.51	1.23	3.5	6.14	10.65

Figura 6.3: *Tabella che raffigura l'accuratezza dei modelli supervisionati*

Dopo aver considerato le *performance* generali del modello è bene considerare anche le metriche relative ad ogni classe. Infatti, ci sono alcune variabili che potrebbero essere state classificate meglio di altre o che hanno ottenuto una miglior punteggio con certi modelli. In tal senso, sono state considerate per ogni classe le seguenti metriche:

- *Recall* (o sensibilità): è una misura di quanto un classificatore sia in grado di identificare correttamente gli elementi di una classe (*True Positive Rate*). Si ottiene come segue:

$$\text{Recall} = \frac{TP}{TP + FN}$$

- *Precision*: è una misura di quanto il classificatore riesce ad identificare correttamente le variabili che non appartengono ad una determinata classe. La precisione indica il rapporto tra il numero delle previsioni corrette sul totale che il modello prevede, in formula:

$$\text{Precision} = \frac{TP}{TP + FP}$$

- *F1-score*: è la media armonica tra la sensibilità e la precisione ed è un buon indicatore della bontà di un modello in quanto considera entrambe le metriche. Ad esempio, potrebbe esserci un modello che identifica benissimo una determinata classe, ma nel farlo commette diversi errori, quindi avrà una sensibilità (*recall*) molto alta ma una precisione bassa. Pertanto lo *f1-score* bilancia questi due aspetti e fornisce un'ottima misura per la bontà dei classificatori;

$$\text{F1 score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

CLASSE 13500 (Fabbri e carpenterie)				
model	CD	NB	SVM	RF
recall	63.12%	80.34%	83.8%	84.14%
precision	84.59%	91.13%	90.11%	88.84%
f1-score	**72.29%**	**85.39%**	**86.85%**	**86.43%**
CLASSE 11000 (Imprese Edili)				
model	CD	NB	SVM	RF
recall	79.59%	84.38%	88.62%	87.83%
precision	77.61%	84.94%	85.01%	84.71%
f1-score	**78.57%**	**84.66%**	**86.78%**	**86.25%**
CLASSE 21100 (Idraulica e riscaldamento)				
model	CD	NB	SVM	RF
recall	73.12%	72.44%	77.8%	75.93%
precision	74.83%	92.43%	86.43%	89.21%
f1-score	**73.97%**	**81.23%**	**81.88%**	**82.04%**
CLASSE 22100 (Elettricisti)				
model	CD	NB	SVM	RF
recall	75.65%	79.10%	81.3%	81.15%
precision	73.16%	83.27%	83.09%	83.62%
f1-score	**74.38%**	**81.13%**	**82.18%**	**82.37%**
CLASSE 42200 (Servizi commerciali)				
model	CD	NB	SVM	RF
recall	72.09%	91.78%	84.70%	86.35%
precision	54.64%	58.68%	69.7%	67.65%
f1-score	**62.17%**	**71.58%**	**76.47%**	**75.87%**

Figura 6.4: *Tabelle che raffigurano recall, precision ed f1-score per ogni classe*

Nella Fig. 6.4 vengono rappresentate le metriche, citate sopra, suddivise sia per categoria che per i diversi modelli di classificazione. Per costruirla sono stati considerati solamente i risultati relativi al *testing* e che consideravano sia unigrammi che bigrammi (2w) con l'aggiunta del codice ateco. In quanto, è sempre preferibile considerare il maggior numero di variabili e come si è visto nella tabella raffigurante l'*accuracy* in Fig. 6.2, fornivano risultati uguali o migliori per tutte le classi.
Dalle tabelle raffigurate si può notare che alcuni classi vengono classificare meglio di altre (come ad esempio le classi 11000 e 13500).

Purtroppo diversi errori di misclassificazione sono dovuti a delle incongruenze nelle *labels* e non sono imputabili ai modelli stessi. Infatti, non sempre il codice Trade aziendale è rappresentativo dei dati provenienti dalla piattaforma Atoka. Questo può essere dovuto al fatto che viene considerata una sola attività economica svolta, quando la ditta del cliente, invece, potrebbe svolgere molteplici attività. Ad esempio un cliente potrebbe essere sia elettricista che idraulico. Questo potrebbe spiegare anche come mai sono presenti più errori di classificazione tra queste due classi, come si può notare in Fig. 6.1.
In tal caso potrebbe risultare utile considerare etichette multiple.

Inoltre, non è da escludere il fatto che alcune incongruenze potrebbero anche essere dovute ad errori di inserimento dei dati. Pertanto, questi modelli potrebbero anche essere adoperati come strumenti di verifica per valutare quali clienti svolgono un'attività economica in linea con quanto presentato ufficialmente in visura camerale.

Capitolo 7

Conclusioni e Sviluppi Futuri

Questo elaborato vuole proporre una soluzione alternativa all'inserimento manuale dei dati. In particolar modo si propone di classificare delle attività economiche, considerando dei codici trade aziendali, sulla base delle loro descrizioni formali.

I risultati ottenuti risultano piuttosto buoni, ma sono ancora lontani per poter considerare la loro ingegnerizzazione ed introduzione nel processo operativo. Difatti questo progetto presenta un caso semplificato e ristretto, dato che si considerano solo le prime 5 attività economiche svolte dai clienti. Inoltre, i dati reali spesso presentano dati incongruenti o mancanti. Una soluzione per arginare i *missing values* potrebbe essere quella di scaricare i dati da Atoka anche di imprese che non sono direttamente clienti di Hilti, ma svolgono quel tipo di mansione riconducibile al codice Trade delle attività. In alternativa, si potrebbero generare dei dati artificiali, tramite API personalizzate ad hoc e *web scraping*. In questo modo si potrebbero estendere l'utilizzo dei classificatori anche per altre attività economiche.

Per quanto riguarda gli algoritmi di classificazione, invece si potrebbero considerare dei modelli ibridi, ossia che combinano approcci basati su regole con i modelli di *Machine Learning*. Ad esempio, nel caso di atti-

vità economiche piuttosto rare si potrebbe costruire una regola univoca, come uno specifico codice ateco, che viene associato ad un determinato codice Trade delle attività aziendali. Sicuramente ridurre il numero e la complessità di codici interni aziendali potrebbe risultare utile ad implementare un sistema automatizzato futuro per il raccoglimento di dati terzi dei clienti.

Si potrebbe considerare anche l'utilizzo di algoritmi di *Deep Learning*, ma senza prima aver attuato degli interventi per migliorare la qualità dei dati non riuscirebbero a dare risultati migliori. Pertanto, prima di scegliere di adottare tecniche più avanzate è bene esplorare i dati che abbiamo a disposizione e gli algoritmi base di ML possono risultare utili per tale intento.

Elenco delle figure

Bibliografia

[1] L. Leslie, "Latex : a document preparation system," 1986.

[2] F. Sebastiani, "Machine learning in automated text categorization," ACM computing surveys (CSUR), vol. 34, no. 1, pp. 1–47, 2002.

[3] M. L. G. Salton, "The smart automatic document retrieval system," 1965.

[4] C. S. Y. G. Salton, A. Wong, "A vector space model for automatic indexing," 1975.

[5] C. B. G. Salton, "Term-weighting approaches in automatic text retrieval," 1988.

[6] A. Samuel, "Some studies in machine learning using the game of checkers," IBM Journal of Research and Development, 1959.

[7] W. S. M. . W. Pitts, "A logical calculus of the ideas immanent in nervous activity," 1943.

[8] A. Turing, "On computable numbers, with an application to the entscheidungsproblem," 1936.

[9] M. Newborn, Kasparov versus Deep Blue: Computer Chess Comes of Age. Springer-Verlag New York Inc, 1997.

[10] Y.-W. T. Geoffrey E Hinton, Simon Osindero, "A fast learning algorithm for deep belief nets,"

[11] Y. L. Y. Bengio, "Convolutional networks for images, speech, and time-series," 1995.

[12] G. E. H. . R. J. W. David E. Rumelhart, "Learning representations by back-propagating errors," pp. 533–536, 1986.

[13] M. R. L. W. Yaniv Taigman, Ming Yang, "Deepface: Closing the gap to human-level performance in face verification," 2014.

[14] T. G. Demis Hassabis, "Mastering the game of go with deep neural networks and tree search," 2016.

[15] M. Minsky, "Steps toward artificial intelligence," pp. 8–23, 1961.

[16] C. C. V. Vladimir, "Support-vector networks," 1995.

[17] "Random forests," 2001.

[18] J. P. J. Han, M. Kamber, Data mining concepts and techniques, third edition. Morgan Kaufmann, 2021.

[19] Istat, Classificazione delle attività economiche Ateco versione 2007, aggiornamento 2022. Istat, 2021.

[20] B. M. Mosley Mark, DAMA-DMBOK: Data Management Body of Knowledge. Technics Publications, 2010.

[21] D. v. L. Anders Haug, Frederik Zachariassen, The costs of poor data quality. Journal of Industrial Engineering and Management, 2011.

[22] W. D. Mack C, Su Z, Managing Missing Data in Patient Registries: Addendum to Registries for Evaluating Patient Outcomes: A User's Guide, Third Edition. Rockville, 2018.

[23] C. Batini and M. Scannapieco, Qualità dei dati: concetti, metodi e tecniche. Springer Science & Business Media, 2008.

[24] L. H. P. K. Nitesh Chawla, Kevin Bowyer, SMOTE: Synthetic Minority Over-sampling Technique. Journal of Intelligence Research, 2002.

[25] T. C. Shilakes, Enteprise Information Portals. Merill Lynch, 1998.

[26] L. H, "Key Word-in-Context Index for Technical Literature (kwic Index).". American Documentation, 1960.

[27] D. H. Roelof van Zwol, Arjen de Vries, Proceedings of the Fifth Dutch-Belgian Workshop on Information Retrieval. Center for Content and Knowledge Engineering Utrecht University, 2005.

[28] I. A. El-Khair, Effects of Stop Words Elimination for Arabic Information Retrieval: A Comparative Study. El-Minia University-Egypt, 2017.

[29] M. F. Porter, An algorithm for suffix stripping. Computer Laboratory Cambridge, 1980.

[30] Y. Goldberg, Neural Network Methods in Natural Language Processing. Neural Network Methods in Natural Language Processing, 2017.

[31] H. P. Luhn, A statistical approach to mechanized encoding and searching of literary information. 1957.

[32] S. Balbi and M. Misuraca, "Pesi e metriche nell'analisi dei dati testuali," Quaderni di Statistica, vol. 7, pp. 55–68, 2005.

[33] S. G., Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. 1989.

[34] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," 1998.